ZW
AF

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

In re Application of: Broussard	§	Group Art Unit: 2173
	§	Examiner: Bonshock, D.
	§	
Serial No. 09/870,620	§	Atty. Dkt. No. AUS920010262US1 (5468-07500)
	§	
Filed: May 31, 2001	§	
	§	
For: SYSTEM AND METHOD FOR FAST DRAWING OF TEXT FIELDS AND LABELS USING A JAVA SWING APPLICATION PROGRAM INTERFACE	§	
	§	
	§	
	§	
	§	

I hereby certify that this correspondence is being transmitted via facsimile or deposited with the U.S. Postal Service with sufficient postage as First Class Mail in an envelope addressed to: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313, on the date indicated below.

2/7/2005
Date

Kevin L. Daffer

APPEAL BRIEF

Mail Stop AF
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313

Sir/Madam:

Further to the Notice of Appeal faxed and received in the U.S. Patent and Trademark Office on December 7, 2004, Appellant presents this Appeal Brief. The Notice of Appeal was filed following mailing of a Final Office Action on September 9, 2004. Appellant hereby appeals to the Board of Patent Appeals and Interferences from a final rejection of claims 1-17 in the Final Office Action, and respectfully requests that this appeal be considered by the Board.

I. REAL PARTY IN INTEREST

The subject application is owned by International Business Machines Corporation, a corporation having its principal place of business at New Orchard Road, Armonk, New York, 10504, as evidenced by the assignment recorded at Reel 011889, Frame 0158.

II. RELATED APPEALS AND INTERFERENCES

Notices of Appeal have been filed for the following applications, which share a common specification with the application currently on appeal.

09/870,614: Notice of Appeal filed 10/26/04; Appeal Brief filed December 20, 2004.
09/870,615: Notice of Appeal filed 9/14/04; Appeal Brief filed November 9, 2004.
09/870,621: Notice of Appeal filed 9/24/04; Appeal Brief filed November 23, 2004.
09/870,622: Notice of Appeal filed 8/24/04; Appeal Brief filed October 25, 2004.

However, because dissimilar art is cited in the present application and the above-mentioned related applications, Appellants do not believe that the outcome of this appeal will have any bearing on the Board's decision on the related appeals. No other appeals or interferences are known which would directly affect or be directly affected by or have a bearing on the Board's decision in this appeal.

III. STATUS OF CLAIMS

Claims 1-16 were originally filed in the present application. Claims 1-4, 7-13, 15, and 16 were amended in a response filed May 13, 2004 to an Office Action mailed February 13, 2004. Claim 17 was added in the same response. Claims 1-17 stand finally rejected under 35 U.S.C. §103, which are the subject of this appeal. A copy of claims 1-17, as on appeal, is included in the Appendix hereto.

IV. STATUS OF AMENDMENTS

No amendments to the claims were filed subsequent to their final rejection. The Appendix hereto therefore reflects the current state of the claims.

V. SUMMARY OF THE INVENTION

Appellant's claimed invention relates to a computer-readable memory or storage device (e.g., memory 18, Fig. 1) and method for drawing an object embedded within software code, which includes text and other displayable content. More specifically, the claimed invention relates to a system and method for fast drawing of text fields and labels using a Java/Swing (referred to as AWTSwing) application program interface. *See*, e.g., Specification, page 32, line 14 to page 34, line 10 and the Title.

In some embodiments, the presently claimed computer-readable memory may include an application program (APP 28, Figs. 1-2) running under an operating system (OS 40, Figs. 1-2), and a first software component, which is adapted to create a graphical representation of an object. The object may be part of a graphical user interface associated with the application program, and may be embodied as code within the first software component. In one aspect of the invention, the code used for creating the graphical representation of the object may include text and other displayable content. In addition to the first software component, the computer-readable memory may further include a second software component, which is adapted for drawing the text that is included within the first software component. As such, the first software component may be invoked during runtime to define visual attributes of the text, without drawing the text, and the second software component may be invoked to draw the text using the visual attributes.

In some cases, the application program may be written in the Java programming language, and the first and second software components may be associated with a Java Virtual Machine (JVM) of the Swing Application Program Interface (API). For example, the first software component may include a Swing JTextField or JLabel component for drawing a graphical representation of a text field or label (i.e., an object) upon a display screen. However, since the first software component is not allowed to draw the text attributable to the object, the second software component (e.g., custom code provided by AWTSwing) may, instead, be used for drawing the text upon the graphical representation of the object. In one aspect of the invention, a peer component (e.g., AWTSwing's JTextFieldPeer or JFastLabelPeer) may be used for redirecting a memory call to invoke text drawing methods of the second software component, rather than text drawing methods of the first software component.

In some embodiments, the presently claimed computer-readable storage device may include a windows-based operating system (OS 40, Figs. 1-2), an application program (APP 28, Figs. 1-2) running under the operating system, an object and text associated with the object, both created at runtime by the application program. In some cases, the application program may be adapted for: (i) invoking a first software component used for creating a graphical representation of the object and defining visual attributes of the object without drawing the text, and (ii) invoking a second software component used for drawing the text using the visual attributes. In order to support undo and redo editing of the text drawn upon the graphical representation of the object, the operating system may place the text to be drawn in a buffer, and the first software component may enable the buffered text to be edited prior to being drawn.

In some embodiments, the presently claimed method for drawing an object embedded within software code, where the object includes text and other displayable content, may include executing a first software component: (i) to create a graphical representation of the object and (ii) to define the visual attributes of the object without creating the text attributable to the object. In some cases, the object may be part of a graphical user interface associated with the application program. For example, the step of executing a first software component may include creating a label upon a display without drawing any text within the boundaries of the label. In other cases, the step of executing a first software component may include creating a border upon the display, where the border represents the outline of a text field, and no text is drawn within the border. The presently claimed method may then execute a second software component to draw the text attributable to the object by using the visual attributes (defined by the first software component) to fetch code that is independent of the operating system. In this manner, the first software component may be used to draw the graphical representation of the object on a display screen, and the second software component may be used to draw the text upon the graphical representation of the object. By utilizing the second software component, the presently claimed method may establish an appearance and behavior of the object that are independent of the operating system.

VI. ISSUES

1. Whether claims 1, 3-10 and 12-17 are unpatentable under 35 U.S.C. §103(a) over a publication written by Matthew T. Nelson, entitled *Java Foundation Classes* (hereinafter "Nelson").

2. Whether claims 2 and 11 are unpatentable under 35 U.S.C. §103(a) over Nelson in view U.S. Patent No. 6,005,588 to Guha (hereinafter "Guha").

VII. GROUPING OF CLAIMS

Claims 1, 3-10 and 12-16 (Group I) stand or fall together.
Claims 2 and 11 (Group II) stand or fall together.
Claim 17 (Group III) stands or falls alone.

The reasons why the three groups of claims are believed to be separately patentable are explained below in the appropriate parts of the Argument.

VIII. ARGUMENT

Swing was developed as part of the Java Foundation Classes in an effort to overcome the platform-dependency of AWT-based (i.e., Java) application programs. An application program interface (API) written using Swing contains no native code, and therefore, can be run on substantially any OS without changing the look and feel of the application. *See*, e.g., Specification, page 9, lines 14-30, and page 19, line 25 to page 20, line 4. Unfortunately, conventional Swing-based APIs lack many desirable features originally included within the AWT implementation. For example, the Swing JTextField and JLabel components often exhibit poorer performance than their AWT TextField and Label counterparts. *See*, e.g., Specification, page 11, lines 1-2.

In other words, conventional Swing-based APIs may suffer performance limitations when using the JTextField and JLabel code to draw text or labels on a display device. The loss in performance may be attributable to one or more of the following factors: (i) Swing's overhead in drawing the text, (ii) the priority level assigned by Swing to the text drawing operation, and (iii) the initialization of undo/redo support. For example, because Swing assigns a relatively low priority level to the text drawing operation, the operating system may frequently suspend text drawing to devote processor cycles to events with higher priority. Each time the processor suspends text drawing to allow another event to execute, it is necessary to preserve the contents of registers, memory locations, etc. containing vital text drawing information, so that the contents may be restored, at a later time, to render the text upon the display. The time lost by frequently saving and restoring the context of the text drawing operation detracts from the performance of text drawing in conventional Swing-based APIs. *See*, e.g., Specification, page 31, line 24 to page 32, line 12.

Therefore, a need exists for an improved system and method that overcomes the performance limitations of the conventional Swing JTextField and JLabel Components.

The invention as recited in claims 1-17 addresses the above-mentioned need by providing two lightweight peer components -- referred to as JTextFieldPeer and JFastLabelPeer -- which may be created for implementing fast text drawing in an enhanced version (referred to as AWTSwing) of the Swing application program interface for Java applications. These lightweight peer components allow the conventional Swing Text Components -- JTextField and JLabel -- to define the "look and feel" of text fields and labels that are displayed in the graphical user interface (GUI) of the Java application.

However, and contrary to the conventional method, the Swing Text Components (i.e., the presently claimed first software components) are not allowed to draw the text. Instead, the lightweight peer components (i.e., JTextFieldPeer and JFastLabelPeer) are used to invoke special fast text drawing code (e.g., embodied within the second software component) for drawing the text. By utilizing the fast text drawing code of the second software component, rather than the drawing code belonging to the JTextField or JLabel components, the present invention allows the "look and feel" of the text field or label component to be determined by Swing, but significantly accelerates the display of text. Once the text is initially drawn, editing functions may be handled by the Swing Text Components in the customary manner. This allows the user to edit the content of the text field or label component with the full support of Swing's editing features. *See*, e.g., the Abstract of the Specification. Reference can also be made to page 32, line 14 to page 34, line 10 of the Specification for further description of the presently claimed system and method, according to one embodiment.

As described in more detail below, none of the cited art, either separately or in combination, provides teaching, suggestion or motivation for the presently claimed computer-readable memory (or storage device) and method for drawing an object embedded within software code, which includes text and other displayable content. More specifically, the cited art fails to disclose (among other things) a software component adapted to create a graphical representation of an object embodied by code, which includes text and other displayable content, and to define visual attributes of the text, without drawing the text. Therefore, the teachings of the cited art cannot be used to render the limitations of the presently claimed case unpatentable.

ISSUE 1 ARGUMENTS

Patentability of Group I Claims 1, 3-10 and 12-16:

1. Nelson fails to provide teaching or suggestion for a first software component that is adapted to: (i) create a graphical representation of an object embodied by code, which includes text and other displayable content, and (ii) define visual attributes of the text without drawing the text.

Independent claim 1 states in part:

A computer-readable memory medium, comprising: a first software component adapted to create a graphical representation of an object embodied as code within the software component, wherein the code comprises text and other displayable content ... wherein the first software

component is invoked during runtime by the application program to define visual attributes of the text, but not to draw the text.

Independent claim 8 (a method) and independent claim 16 (a computer-readable storage device) recite similar limitations.

Nelson provides an overview of the Java Foundation Classes, which include Swing classes and others (Nelson, page xxv). Nelson, however, does not teach or suggest a first software component that is adapted to: (i) create a graphical representation of an object embodied by code, which includes text and other displayable content, and (ii) define visual attributes of the text without drawing the text. Statements in the Final Office Action suggest that teaching for the presently claimed first software component may be found on pages 694, 697 and 78 of Nelson (*See*, e.g., Final Office Action, pages 2-3). The Appellant respectfully disagrees, and traverses the Examiner's assertion of teaching on a point-by-point basis in more detail below.

On page 694, Nelson introduces the Swing JTextField Component and notes that the component may be used for getting simple text input and editing single-line text strings. Contrary to the statements in the Final Office Action, however, the Swing JTextField Component cannot be used to provide teaching or suggestion for the presently claimed first software component. For example, the Swing JTextField Component cannot be used for defining the visual attributes of a text string associated with a text field, without also drawing the text string if it is specified. If the text string is not specified, the JTextField Component may be used to create a new text field without drawing any text. However, such a case is inconsistent with the presently claimed first software component, which specifically includes "text and other displayable content" within the object code of the first software component. In order for the JTextField Component to read upon the presently claimed first software component, there must be some teaching or suggestion within Nelson for not drawing any text when a text string (e.g., "Horatio Hornblower", as shown on page 694 of Nelson) is specified for the JTextField Component. Such teaching or suggestion simply cannot be found on page 694, or anywhere else within the teachings of Nelson.

On page 697, Nelson describes how a border of the Swing JTextField Component may be changed by using the "setBorder" and "setBackground" methods. Though the "setBorder" and "setBackground" methods may be used to change the "look" of the Swing JTextField Component (e.g., by selecting a different type of border or background color), the methods cannot be used either

separately, or in conjunction with the JTextField Component, to provide teaching or suggestion for the presently claimed first software component. For example, if the "setBorder" and "setBackground" methods are considered separately, the methods do not provide teaching or suggestion for the presently claimed first software component, since neither the "setBorder" nor the "setBackground" method is adapted to create a graphical representation of an object. Instead, and as noted above, the setBorder" and the "setBackground" methods are merely used to select a type of border and a background color, respectively. Even if the "setBorder" and "setBackground" methods were combined with the Swing JTextField Component, the combination of software components and methods would still fail to provide teaching or suggestion for the presently claimed first software component. Since Nelson fails to teach or suggest that the JTextField Component may refrain from drawing text when a text string is specified, and the "setBorder" and "setBackground" methods have absolutely nothing to do with drawing text, the software components and methods cannot be combined in such a way to provide the presently claimed first software component and/or the functionality thereof. As a consequence, absolutely no teaching or suggestion can be found on page 697 of Nelson for the presently claimed first software component.

On page 3 of the Final Office Action, the Examiner suggests that "Nelson further teaches on page 78, UI class having separate groups of code to get the look-and-feel, and to draw the text... [for example] it teaches a UI class that doesn't know what the text control contains or what the contents should look like, but uses getDocument() and getStyledDocument() methods." The Examiner's suggestions are repeated on page 7 of the Response to Arguments section of the Final Office Action. In the above Office Action statements, the Examiner appears to suggest that a Swing UI class may be used to "get the look-and-feel" of a Text Component, while the getDocument() and getStyledDocument() methods may be used for drawing the text attributable to the Text Component. Therefore, the Examiner appears to rely on a Swing UI class to provide teaching for the presently claimed first software component and the getDocument() and getStyledDocument() methods to provide teaching for the presently claimed second software component. The Appellant respectfully disagrees with the Examiner's suggestions, as set forth in more detail below.

Contrary to the Examiner's statements, the getDocument() and getStyledDocument() methods are not used for drawing the text attributable to a Text Component, but instead, are used merely for accessing or fetching a document (or styled document) associated with the Text Component. These methods are embodied within the Text Component, and may be used by a Swing UI class to get the document (or styled document) when it is time for the Swing UI class to draw the Text Component. *See*, e.g., Nelson,

page 78, under the heading of "Interacting with Text Components". Since the getDocument() and getStyledDocument() methods (as described on page 78 of Nelson) are not used for actually drawing text, the methods cannot be considered equivalent to the presently claimed second software component. The Examiner's implied suggestion of such is hereby traversed.

In addition, though a Swing UI class may be used to "get the look-and-feel" of a Text Component, a Swing UI class cannot be considered equivalent to the presently claimed first software component, as recited in claims 1 and 8. For example, a Swing UI class does not contain code, which includes text and other content to be displayed, as clearly specified in the presently claimed first software component. Instead of including text and other displayable content, the Swing UI class (or "Look-and-Feel class") of Nelson simply includes "a reference back to the Text Component" (*See*, Nelson, page 78, emphasis added). A "reference back to a Text Component" cannot be considered equivalent to the actual text, itself.

For the UI class of Nelson to read upon the first software component, the UI class would have to include the presently claimed "code comprising text and other displayable content", as well as the visual attributes to be used in displaying the text, without including additional code for drawing the text. As noted above, the UI class of Nelson does NOT include "code comprising text and other displayable content." In addition, Nelson specifically states that the UI class includes "code for drawing the [text] component ..." and "is responsible for drawing the [text] component" (Nelson, page 78). Since the UI class of Nelson is responsible for drawing the text (*See*, Nelson, page 78, under the heading of "Interacting with Text Components"), the UI class of Nelson cannot be considered equivalent to the presently claimed first software component, as apparently suggested by the Examiner.

On page 8 of the Response To Arguments section of the Final Office Action, the Examiner states, "[t]he applicant's argue that the JtextField component cannot be used for defining the visual attributes of a text string without also drawing the text string. In response, the examiner respectfully submits that Nelson teaches on page 695 that there is a constructor that is able to transfer a complete document (with defined text) from [one] text component to another." The Appellants are confused as to how a "constructor that is able to transfer a complete document from one text component to another" can be used to provide teaching for the presently claimed first software component. Transferring a complete document from one text component to another is not a claimed feature of the invention.

The Examiner further states, "[i]t is further shown that pages 73 and 75 [of Nelson] teach the limitation of one software component for defining and one software component for drawing the actual text, [regardless of] whether or not these components are JtextField and JLabel or not. The applicant has further admitted on page 11 of their response [to the Office Action mailed May 13, 2004] that separate groups of swing based code can be used for setting the look and feel of a displayed object and for drawing text within or upon the displayed object." (Final Office Action, page 8). In the above Office Action statements, it appears that the Examiner is attempting to find some teaching within Nelson (or Applicant's previous remarks) by interpreting the claim language more broadly than it reasonably allows. However, during patent examination, the pending claims must be "given the broadest reasonable interpretation consistent with the specification." MPEP 2111.

Present claims 1, 8 and 16 recite limitations on a first software component, which is adapted to create a graphical representation of an object (which is embodied as code including text and other displayable content), and a second software component adapted for drawing the text. More specifically, the presently claimed first software component may be invoked during runtime to define visual attributes of the text, without drawing the text, whereas the second software component may be invoked to draw the text using the visual attributes. By interpreting the invention to include "one software component for defining and one software for drawing the actual text," the Examiner improperly broadens the limitations actually recited in claims 1, 8 and 16. Appellant's assert that the claims cannot be interpreted so broadly, so as to enable portions of the claim limitations to be overlooked during examination. For example, one cannot ignore the fact that the presently claimed first software component includes object code comprising text and other displayable content, as recited in present claims 1 and 8. As noted above, the Swing UI class disclosed, e.g., on page 78 of Nelson does not include the actual text (or "Document") associated with the Text Component, and therefore, cannot be considered equivalent to the presently claimed first software component. In providing such a loose interpretation of the presently claimed case, the Examiner fails to provide sufficient evidence within Nelson for the limitations actually recited in present claims 1, 8 and 16.

2. There is no motivation to modify the teachings of Nelson to provide the presently claimed first software component.

The Appellant also wishes to traverse, or otherwise correct, another statement that was made in the Final Office Action. On page 3 of the Final Office Action, the Examiner suggests that it "would have been obvious to one of ordinary skill in the art, having the teachings of Nelson that the JtextField and

Jlabel components could be combined, to allow for one [of the] components to handle look-and-feel and one component to handle the displaying of the text."

First of all, Nelson provides absolutely no teaching, suggestion, motivation or even desirability for combining the JTextField and JLabel components. Obviousness can only be established by combining or modifying the teachings of the prior art to produce the claimed invention where there is some teaching, suggestion, or motivation to do so. *In re Fine*, 837 F.2d 1071, 5 USPQ2d 1596 (Fed.Cir. 1988); *In re Jones*, 958 F.2d 347, 21 USPQ2d 1941 (Fed. Cir. 1992); MPEP 2143.01. The combination proposed by the Examiner simply cannot be made since no teaching, suggestion or motivation exists within Nelson.

Furthermore, even if the JTextField and JLabel components were somehow combined (without sufficient motivation to do so), the proposed combination would not only fail to read upon the present claims, but would also change the principle operation of the JTextField and JLabel components. If proposed modification or combination would change the principle of operation of the prior art invention being modified, then the teachings of the reference is not sufficient to render the claims *prima facie* obvious. *In re Ratti*, 270 F.2d 810, 123 USPQ 349 (CCPA 1959).

For example, although the JTextField and JLabel components can each be used for displaying text on a display screen, the JTextField component is used for displaying editable text, whereas the JLabel component is used for displaying static text, and thus, has no editing capability. *See*, e.g., Specification, page 33, lines 3-6. If the JTextField and JLabel components were somehow combined (as suggested by the Examiner), it is conceivable that the JTextField component could lose it's editing capability, or the JLabel could gain editing capability where none is desired, thereby changing the principle operation of the components. Regardless, Appellants assert that the JTextField and JLabel components cannot be combined in a manner that would read upon the presently claimed first and second software components, as they are actually recited in claims 1, 8 and 16. As noted above, suggesting that the Swing JTextField and JLabel components could be combined "to allow for one [of the] components to handle look-and-feel and one component to handle the displaying of the text," is an unreasonably broad interpretation of claim limitations.

3. The Examiner has failed to adequately support and/or establish a *prima facie* ground of obviousness.

To establish a prima facie case of obviousness, three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) must teach or suggest all claim limitations. MPEP § 2143. None of these three criteria have been met by the Examiner in the present case. First of all, no suggestion or motivation to modify the teachings of Nelson can be found within Nelson to teach or suggest the aforementioned limitations of claims 1, 8 and 16, as explained above in Argument 2. The criterion of a reasonable expectation of success cannot be met if no teaching, suggestion or motivation exists, because there is then nothing at which to be successful. Finally, Nelson fails to teach all of the limitations of claims 1, 8 and 16, as explained above in Argument 1. The third criterion recited above has therefore also not been met, and a *prima facie* case of obviousness has not been established.

Conclusion

As explained in Arguments 1-3 above, at least some limitations of claims 1, 8 and 16, and therefore, at least some limitations of claims 2-7, 9-15 and 17, are not taught or suggested by the cited art. Furthermore, there is no teaching, suggestion or motivation to modify the cited art to teach the limitations of these claims. For at least the reasons set forth above, claims 1-17 are patentably distinct over the cited art. Therefore, the rejection of Group I claims 1, 3-10 and 12-16 under 35 U.S.C. §103(a) is asserted to be erroneous.

Patentability of Group III Claim 17:

Because claim 17 of Group II is dependent from claim 1, the arguments presented above for patentability of Group I claims 1, 3-10 and 12-16 apply equally to claim 17, and are herein incorporated by reference. In addition to the arguments presented above with respect to claims 1, 3-10 and 12-16, arguments are provided below to establish patentability of the current claim under 35 U.S.C. § 103(a).

1. **Nelson fails to provide teaching, suggestion or motivation for a peer component, which is adapted for redirecting a call to invoke text drawing methods of the second software component rather than the text drawing methods of the first software component.**

Claim 17 states, in part, "[t]he memory ... further comprising a peer component adapted for redirecting a ... call to invoke text drawing methods of the second software component rather than text drawing methods of the first software component." For example, and as noted above, the presently claimed case provides lightweight peer components (i.e., JTextFieldPeer and JFastLabelPeer) for redirecting calls, initially intended to invoke the text drawing methods of the Swing JTextField and JLabel components, to an internal fast text drawing routine embodied within the presently claimed second software component. *See*, e.g., Specification, page 32, lines 14-28 and page 33, line 14 to page 34, line 2.

In addition to failing to disclose the presently claimed first and second software components, Nelson fails to provide teaching, suggestion or motivation for a peer component, which redirects a call to invoke the text drawing methods of the second software component rather than the text drawing methods of the first software component. However, the Examiner suggests that teaching for the presently claimed peer component can be found within Nelson. For example, the Examiner states, "Nelson teaches, on pages 694, 697, 472, and on page 72, a system for drawing text where one component can define attributes of an item and the actual displaying of the item can be implemented by another item." (Final Office Action, page 6). The Appellant respectfully disagrees, as described in more detail below.

First of all, "a system for drawing text where one component can define attributes of an item and the actual displaying of the item can be implemented by another item," is not equivalent, nor can it be interpreted to read upon the presently claimed peer component. In addition, the Appellants assert that no teaching, suggestion or motivation for the presently claimed peer component -- which redirects a call to invoke the text drawing methods of the second software component rather than the text drawing methods of the first software component -- can be found within the pages cited by the Examiner, or anywhere else within the teachings of Nelson. As such, Nelson cannot be relied upon to provide teaching or suggestion for the limitation recited in present claim 17.

2. There is no motivation to modify the teachings of Nelson to provide the presently claimed peer component.

Obviousness can only be established by combining or modifying the teachings of the prior art to produce the claimed invention where there is some teaching, suggestion, or motivation to do so. *In re Fine*, 837 F.2d 1071, 5 USPQ2d 1596 (Fed.Cir. 1988); *In re Jones*, 958 F.2d 347, 21 USPQ2d 1941 (Fed. Cir. 1992); MPEP 2143.01. Absent any teaching, suggestion, or motivation to do so, Appellants assert that the teachings of Nelson cannot be modified to provide teaching for the presently claimed peer component, as recited in claim 17.

3. The Examiner has failed to adequately support and/or establish a p*rima facie* ground of obviousness.

To establish a prima facie case of obviousness, three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) must teach or suggest all claim limitations. MPEP § 2143. None of these three criteria have been met by the Examiner in the present case. First of all, no suggestion or motivation to modify the teachings of Nelson can be found within Nelson to teach or suggest the aforementioned limitations of claim 17, as explained above in Argument 2. The criterion of a reasonable expectation of success cannot be met if no teaching, suggestion or motivation exists, because there is then nothing at which to be successful. Finally, Nelson fails to teach all of the limitations of claim 17, as explained above in Argument 1. The third criterion recited above has therefore also not been met, and a *prima facie* case of obviousness has not been established.

Conclusion

As explained in Arguments 1-3 above, the limitation recited in claim 17 is not taught or suggested by the cited art. Furthermore, there is no teaching, suggestion or motivation to modify the cited art to teach the limitation of that claim. For at least the reasons set forth above, claim 17 is patentably distinct over the cited art. Therefore, the rejection of Group III claim 17 under 35 U.S.C. § 103(a) is asserted to be erroneous.

ISSUE 2 ARGUMENTS

Patentability of Group II Claims 2 and 11:

Because claims 2 and 11 of Group II are dependent from claims 1 and 8, respectively, the arguments presented above for the patentability of claims 1 and 8 apply equally to claims 2 and 11, and are herein incorporated by reference. In addition to the arguments presented above with respect to claims 1 and 8, arguments are provided below to establish patentability of the current claims under 35 U.S.C. § 103(a).

Guha cannot be combined with Nelson to provide teaching or suggestion for the first software component, as recited in present claims 1 and 8.

The secondary reference to Guha is not cited against present claims 1 and 8. However, the Appellant wishes to point out that since Guha also fails to provide teaching or suggestion for the presently claimed first software component, Guha cannot be combined with Nelson to overcome the deficiencies therein. In other words, the combination of Guha and Nelson would still fail to provide teaching or suggestion for the presently claimed first software component, as recited in claims 1 and 8.

Conclusion

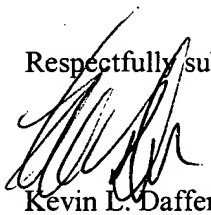
As explained in the Argument above, Guha and Nelson cannot be combined to teach or suggest all limitations of claims 1 and 8. Furthermore, there is no teaching, suggestion or motivation to modify the cited art to teach the limitations of those claims. Since claims 2 and 11 are dependent on claims 1 and 8, respectively, claims 2 and 11 are patentably distinct over the cited art for at least the same reasons as claims 1 and 8. Therefore, the rejection of Group II claims 2 and 11 under 35 U.S.C. § 103(a) is asserted to be erroneous.

IX. CONCLUSION

For the foregoing reasons, it is submitted that the Examiner's rejection of claims 1-17 was erroneous, and reversal of the Examiner's decision is respectfully requested.

The Commissioner is hereby authorized to charge the required fee(s) to deposit account number 50-3268/5468-07500.

Respectfully submitted,



Kevin L. Daffer
Reg. No. 34,146
Attorney for Appellant

Daffer McDaniel, LLP
P.O. Box 684908
Austin, TX 78768-4908
Date: February 7, 2005

X. APPENDIX

The present claims on appeal are as follows.

1. A computer-readable memory medium, comprising:

a first software component adapted to create a graphical representation of an object embodied as code within the software component, wherein the code comprises text and other displayable content;

an application program running under an operating system; and

a second software component adapted for drawing the text, wherein the first software component is invoked during runtime by the application program to define visual attributes of the text, but not to draw the text, and wherein the second software component is invoked to draw the text using the visual attributes.

2. The memory as recited in claim 1, wherein the operating system places the text to be drawn in a buffer, and wherein the first software component enables the buffered text to be edited prior to being drawn.

3. The memory as recited in claim 1, wherein the first software component is used for drawing the graphical representation of the object on a display screen, and wherein the second software component is used for drawing the text upon the graphical representation of the object.

4. The memory as recited in claim 3, wherein the first software component is adapted to support undo and redo editing of the text drawn upon the graphical representation of the object.

5. The memory as recited in claim 1, wherein the object is part of a graphical user interface associated with the application program.

6. The memory as recited in claim 1, wherein the application program is written in Java programming language.

7. The memory as recited in claim 1, wherein the first and second software components comprises a Java virtual machine of a Swing application program interface.

8. A method for drawing an object embedded within software code, said object comprising text and other displayable content created by an application program running under an operating system, the method comprising:

executing a first software component to create a graphical representation of the object to define the visual attributes of the object without creating the text attributable to the object; and

executing a second software component to draw the text attributable to the object by using the visual attributes to fetch code that is independent of the operating system.

9. The method as recited in claim 8, wherein said executing a first software component comprises creating a label upon the display absent any text.

10. The method as recited in claim 8, wherein said executing a first software component comprises creating a border upon the display absent any text within the border.

11. The method as recited in claim 8, further comprising the operating system placing the text to be drawn in a buffer, and wherein said executing a first software component comprises editing the buffered text prior to the buffered text being drawn.

12. The method as recited in claim 8, further comprising using the first software component to draw the graphical representation of the object on a display screen, and using the second software component to draw the text upon the graphical representation of the object.

13. The method as recited in claim 12, further comprising the first software component supporting undo and redo editing of the text drawn upon the graphical representation of the object.

14. The method as recited in claim 8, wherein the object is part of a graphical user interface associated with the application program.

15. The method as recited in claim 8, further comprising using the second software component to establish an appearance and behavior of the object that are independent of the operating system.

16. A computer-readable storage device, comprising:

a windows-based operating system;

an application program running under the operating system;

an object and text associated with the object, both created at runtime by the application program, wherein the application program is adapted for:

invoking a first software component adapted for creating a graphical representation of the object and defining visual attributes of the object without drawing the text; and

invoking a second software component adapted for drawing the text using the visual attributes.

17. The memory as recited in claim 3, further comprising a peer component adapted for redirecting a memory call to invoke text drawing methods of the second software component rather than text drawing methods of the first software component.